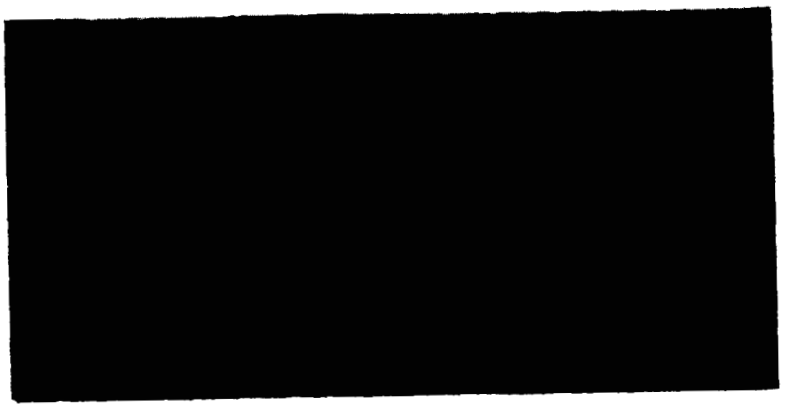
NAS2-11530



(NASA-CR-187275) EXPERT SYSTEMS (Research Inst. for Advanced Computer Science) 16 p

N90-71380

Unclas
00/62 0295401

RIACS

Research Institute for Advanced Computer Science

Expert Systems

Peter J. Denning

November 8, 1985

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 85.17

Expert Systems

Peter J. Denning

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 85.17
November 8, 1985

Computer programs that explicitly represent the knowledge of experts can be used to solve problems normally handled by those experts. These programs are subject to the same limitations on their capability as other software. The heuristic techniques used to program them are likely to lead to systems of less reliability and maintainability than programs developed with precise specifications.

Work reported herein was supported in part by Contract NAS2-11530 from the National Aeronautics and Space Administration (NASA) to the Universities Space Research Association (USRA).

This is a preprint of the column *The Science of Computing* for *American Scientist 74*, No 1, January-February, 1986.

Expert Systems

Peter J. Denning
Research Institute for Advanced Computer Science

November 8, 1985

We humans have held high opinions of ourselves throughout the ages. We define ourselves, *Homo sapiens*, as the only earthly creatures capable of rational thought. Perhaps as expressions of a racial urge to be godlike, we aspire not only to control our environment, but to create artificial beings.

In *Machines Who Think*, Pamela McCorduck gives a fascinating account of how this aspiration has shown itself down through history *(1)*. She sees the urge to create artificial beings expressed in ancient idols; in medieval legends of homonculi or golems; in the calculating machines, mechanical statues, and chess automatons of later centuries; and in the current quest for thinking machines. I might add that the urge also expresses itself in genetic engineering.

Our creative urge is coupled with a tremendous reverence for logic. The idea that the ability to reason logically -- to be rational -- is closely tied with intelligence was clear in the writings of Plato. The search for greater understanding of human intelligence led to the development of mathematical logic, the

study of methods of proving the truth of statements by manipulating the symbols in which they are written without regard to the meanings of those symbols. By the nineteenth century a search was under way for a universal logic system, one capable of proving anything provable in any other system.

This search came to an end in the 1930s with Kurt Godel's incompleteness theorem and Alan Turing's incomputability theorem. Godel showed that, given any sufficiently powerful, consistent system of logic, one can construct a true proposition that cannot be proved within that system. Turing described a universal computer and showed that there are well defined problems that cannot be solved by any computer program, even in principle. As an example, he showed that there is no program that can determine whether any other program will enter an endless loop when executed. A consequence of these results is that there are always theorems beyond logic and tasks beyond machines. Some philosophers argue that human intelligence is higher: there will always be human accomplishments that defy logic or mechanical simulation.

Despite the fact that logical deductions can be carried out by machines as pure symbol manipulations without regard to content, we continue to associate logic with intelligence. Indeed, the fact that this intelligent activity is well defined, reliable, and mechanizable appeals strongly to the urge to create a thinking machine. One cannot fully appreciate modern research in machine intelligence, its frequent psychological allusions and occasional mysticism, without understanding the human urges to create life and to explain intelligence

with logic. Knowing this, one can peer past the aura to the reality.

The term artificial intelligence (AI) was first used in 1956 by John McCarthy of Stanford University. It refers to the subfield of computer science that studies how machines might behave like people. Several technologies popularly associated with AI have reached the marketplace -- notably image pattern recognition, speech recognition, speech generation, and robotics. However, the commercially successful versions of these technologies do not exploit symbolic manipulations, and they were built by engineers who do not profess special skill in AI. But one technology in particular has captured many fancies: expert systems.

An expert system is a computer system designed to simulate the problem-solving behavior of a human who is expert in a narrow domain. Examples include prescribing antibiotics, classifying chemical compounds, configuring and pricing computer systems, scheduling experiments in real time for a telescope, classifying patterns in images, and diagnosing equipment failures. Like books, expert systems are a way to make the knowledge of a few available to the many.

In mathematical logic, a system consists of (finite) sets of axioms and rules of inference. A proof is a sequence of strings of symbols, say $S_1, \ldots, S_p$, such that each string S_i either is an axiom or is derivable from some subset of $S_1, \ldots, S_{p-1}$ by a rule of inference. An expert system is a restricted form of a system of logic. It attempts to compute a sequence of strings representing the steps in the solution of a problem. The sequence serves as a proof of the

solution. The rules of inference are of a simple form that represents a common pattern in problem-solving: "if *conditions* then *consequences*."

An expert system has three main parts: a database for storing axioms and rules of inference; an algorithm for constructing proofs; and a user interface. The interface, which often includes powerful interactive graphics, provides a language for the user to express queries and to provide information to the system. In expert-system jargon, axioms are called "facts," rules of inference "rules," the database the "knowledge base," the database programmer the "knowledge engineer," the proof-constructing algorithm the "inference engine," proof-construction "reasoning," and a proof of a solution an "explanation."

A number of companies now market expert systems "shells" that come with empty knowledge databases. Where do the data come from? A database is built by a trial-and-error process called knowledge engineering. Through extensive interviews with the expert, the knowledge engineer attempts to elicit verbalizations of rules and of highly detailed facts, and to program this information into the database. The process includes continual testing of the system and expansion of its database to cover cases previously left out.

Although prototypes of systems based on a few hundred rules can often be brought into operation within two months, it is not unusual for the process of building and testing an expert system to require many months and to produce several thousand rules. The set of rules may be incomplete and inconsistent; the exact behavior of the system may have no precise specification and may not be

reliably predictable by its designer. For this reason, the technique of programming an expert system is sometimes called heuristic programming.

This class of systems using heuristic programming and rule evaluation does not contain all systems that simulate human expertise. John Shore points out that autopilots -- control systems that solve the equations of motion of the aircraft -- fly planes very well. They are expert fliers, but not expert systems *(2)*.

AI researchers often distinguish between shallow and deep expert systems. Shallow systems are designed for speed. In their limited databases they store more facts than rules. The proofs of their conclusions are usually short, and most of the conclusions strike observers as being straightforward consequences of information in their databases. They often give poor results when applied to problems other than those for which they have been tested. Examples include a system to generate production schedules using bin-packing heuristics and a system to answer electronic inquiries for technical information.

Deep systems derive their conclusions from models of phenomena in their domains, using first principles embedded in the model. The proofs of their conclusions are usually long, and some conclusions may strike some observers as anything but obvious. An example is Dendral, a system that, when given spectral data, generates diagrams of chemical compounds using ball-and-stick atomic models. Other prominent examples include qualitative physics programs, which determine states of behavior of physical systems *(3)*. These programs can proceed backwards from observations and generate hypotheses about initial

conditions; they can proceed forward from initial conditions and explain discrepancies between predicted and observed final states by generating hypotheses about which components may be broken.

The distinction between shallow and deep systems is not sharp. It may not even be important. A system that derives conclusions from first principles may generate long proofs, but how long is a deep proof? Every expert system is capable of generating conclusions that are not obvious to some observers, but what fraction of observers will be surprised by a deep system? A model is a small set of rules that explains a large amount of data, but how much compaction of data into rules is deep?

A system need not be deep to be useful. One of the best-known expert system is Mycin, whose rules associate diseases with symptoms. By answering questions about symptoms, Mycin leads the user to a diagnosis and recommended antibiotic treatment. The processes by which Mycin reaches its conclusions resemble those used by a doctor in interviewing a patient and analyzing lab data. Mycin is not a deep system, because its knowledge is empirical and is not derived internally from models of diseases, symptoms, or cures.

Every expert system, deep or shallow, is likely to use one of three ways of representing knowledge. The three resulting types of system are nicely described in three articles edited by Peter Friedland of Stanford University *(4)*. I will summarize these systems briefly here.

The first type is the logic programming system. It embodies explicitly the model of proof in mathematical logic. In the language PROLOG, for example, rules are represented in the form "q :- $p_1, \ldots, p_n$," interpreted as: "If the predicates $p_1, \ldots, p_n$ are all true, then the predicate q follows." A predicate is a formula that is true for some values of its variables; for example, PARENT(x,y) is true exactly when person x is the parent of person y. A fact is represented by an empty right side; for example, if Alice is the parent of Bob, we write "PARENT(Alice, Bob) :-." The principle of transitivity allows us to replace a pair of rules,

$$Q \text{ :- } p_1, \ldots, p_n$$

$$s \text{ :- } Q, r_1, \ldots, r_m$$

with the new rule

$$s \text{ :- } p_1, \ldots, p_n, r_1, \ldots, r_m .$$

This pattern of reduction is the basis of the PROLOG inference algorithm, called resolution, for finding the ultimate consequences of given conditions. The resolution principle is actually much more powerful than described above. It is not necessary that the Q on the left of the first rule be the same as the Q on the right of the second rule; it is only necessary that they can be made the same by substituting for some of the variables. For example, an allowable PROLOG resolution takes

PARENT(x,Bob) :- MOTHER(x,Bob)

GRANDPARENT(Alice,z) :- PARENT(Alice,y), PARENT(y,z)

into

GRANDPARENT(Alice,z) :- MOTHER(Alice,Bob), PARENT(Bob,z)

after assigning x =Alice and y =Bob. The process of forcing two predicates to be the same by finding matches among components is called unification. With each resolution, PROLOG keeps track of the possible values that make the predicates true. Not only can it answer queries with a "yes" or "no," it can report which values variables must have for "yes" to be the answer.

The second type is the rule-based system. Its database is structured as a hierarchy of rule sets. Each rule set comprises a series of condition-consequence rules as defined earlier. Two strategies are common for evaluating the consequences of the initial conditions given by the user. Under forward chaining, evaluation proceeds by tracing rules from left side to right, until the ultimate consequences of the given conditions have been found. Under backward chaining, evaluation proceeds by tracing rules from right side to left, until the ultimate antecedents of a hypothesis about the given conditions have been found; hypotheses that conflict with the given conditions are discarded along the way. Which method is faster depends on the problem. If many data are available, it is often faster to chain backwards from a few likely hypotheses. If there are many possible hypotheses (for example, in medical diagnosis), it is usually better

to chain forward from the limited initial data. The search for applicable rules can be restricted by allowing consequents of rules to name rule-sets in which further applicable rules can be found.

The third type is the frame-based system. A frame based system is centered on a hierarchy of descriptions of objects referred to in the rules. The description of a class of objects or an individual object is called a frame. The relation "is an instance of" organizes the hierarchy of frames. Thus in a computer failure diagnosis system, a frame called "Microprocessor No. 3" might be immediately subordinate to a frame called "Microcomputers," which in turn might be subordinate to a frame called "Computers." Rules are used to describe functional relationships among frames. Any property of an object that can be derived by locating it in the hierarchy of frames need not be explicitly represented in the rule sets. This gains relative simplicity of rule sets in exchange for complexity of the catalogs of frames and of the inference algorithm (which must do more computation).

What are the limitations of expert systems? The most important concern their reliability. Expert systems are limited by the information in their databases and by the nature of the process for putting that information in. They cannot report conclusions that are not already implicit in their databases. The trial-and-error process by which knowledge is elicited, programmed, and tested is likely to produce inconsistent and incomplete databases; hence, an expert system may exhibit important gaps in knowledge at unexpected times. Moreover, expert

systems are unlikely to have complete, clear functional specifications, and their designers may be unable to predict reliably their behavior in situations not tested. If not carefully structured as modules, large databases are likely to be difficult to modify and maintain.

This aspect of expert systems has been sharply criticized. David Parnas argues that software developed heuristically is inherently less reliable than software developed from precise specifications *(5)*. He asserts that to get a reliable program to solve a problem one must study the problem, not the way people say they solve it. For example, to distinguish among objects in a picture, one studies the characteristics of the objects and of the photographic process; if one asks people for the rules, one is unlikely to find a reliable program. The expert may not be consciously aware why his methods work, and the knowledge engineer who interviews him is like an investigative reporter who may not always ask the right questions.

John Shore reminds us that testing is an inherently unreliable way to find errors in programs *(2)*. Because it can only reveal the presence of bugs, never their absence, testing is inadequate for most software. The situation is worse for expert systems, where the correct behavior in a particular case is often a matter of opinion. In Shore's view, programming by trial and error will produce expert systems that are unreliable and cannot be trusted as autonomous systems. He does believe that expert systems can be useful, with human supervision, as "intelligent assistants."

Some AI researchers respond that complete testing is meaningless for expert systems. The best these systems can do is imitate human experts, who are themselves imperfect. Even as one does not expect a doctor to diagnose correctly every illness, one cannot expect an expert system to do better. They respond further that mathematical proof is meaningless for large programs because one cannot tell whether the specifications are complete or accurate. While I find much to agree with in these statements, I do not support their implied conclusion. I believe that we can and often do build machines that overcome human limitations. I believe further that in producing machines on whose decisions lives may depend we should use every available method of establishing that the machines will perform their functions reliably. While imperfect, precise specifications are more likely to produce a reliable machine than are trial and error.

Insistence by their developers that expert systems cannot be made more reliable than humans will be met by public insistence that software pass tests similar to those for certifying human experts. This may significantly extend the time required to bring an expert system to market. Other tests may be imposed too. For example, the Food and Drug Administration is considering a requirement that medical diagnosis systems be subjected to stringent field tests like other medical apparatus before being allowed on the general market.

Although expert systems developed by trial and error are likely to be unreliable, there is no reason in principle that, with a different programming methodology, expert systems cannot be as reliable as other software. Expert systems

are, after all, nothing more than computer programs. The powerful languages of logic, rules, and frames permit deductive processes to be programmed quite rapidly. Like other software, there is every reason to believe that rules can be constructed as sets of modules, that modules can be organized into hierarchies, that calculi for precise specifications can be developed, and that methods for proving that rules meet specifications can be devised.

The notion of precise specifications for human processes has its quixotic side, for how does one specify precisely what an expert does? And yet there are grounds for hope. The most successful expert systems to date are the ones used for diagnosis. They are based on a well-understood model of diagnostic processes. The rules can be organized into a hierarchy of modules corresponding to decision points in the diagnostic process; their completeness and consistency can be evaluated with respect to the model. The hope is that models of other human problem-solving processes can be defined and used similarly for their expert systems.

There is nothing magic about expert systems. Turing's thesis tells us that any mechanical procedure can be programmed on a computer. If we know of a procedure to find a procedure, we can program that. But if we know nothing at all, we are stuck. We cannot expect an expert system to help if we do not know how something is done. Alan Perlis of Yale has said: "Good work in AI concerns the automation of things we know how to do, not the automation of things we would like to know how to do." AI cannot replace RI.

References:

1. P. McCorduck. 1979. *Machines Who Think.* Freeman.

2. J. E. Shore. 1985. *The Sachertorte Algorithm.* Viking.

3. D. Bobrow, ed. 1985. *Qualitative Reasoning About Physical Systems.* MIT Press.

4. P. Friedland, ed. 1985. Knowledge-based architectures. *Communications of ACM*, September.

5. D. L. Parnas. 1985. "Software Aspects of Strategic Defense Systems." *American Scientist 73*, 5, September-October.